AD-A034 480 WISCONSIN UNIV MADISON MATHEMATICS RESEARCH CENTER F/G 12/1
APPLICATION OF INTERVAL ANALYSIS TO ERROR CONTROL.(U)
SEP 76 J M YOHE DAAG29-75-C-0024
UNCLASSIFIED MRC-TSR-1686 NL

1 OF 1
AD A034480

END

DATE
FILMED
2 - 77

FG

13

MRC Technical Summary Report #1686

APPLICATION OF INTERVAL ANALYSIS TO ERROR CONTROL

J. M. Yohe

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53706

September 1976

Received August 30, 1976

Approved for public release
Distribution unlimited

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

and

Waterways Experiment Station
Vicksburg, Mississippi 39180

UNIVERSITY OF WISCONSIN - MADISON
MATHEMATICS RESEARCH CENTER

APPLICATION OF INTERVAL ANALYSIS TO ERROR CONTROL

J. M. Yohe

Technical Summary Report #1686
September 1976

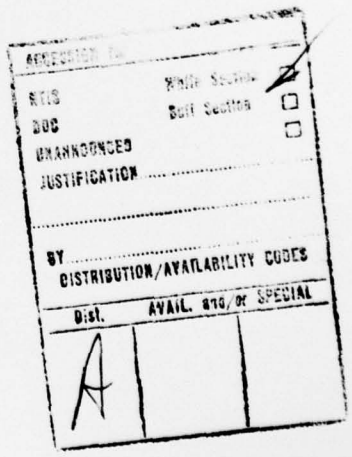
ABSTRACT

We give simple examples of ways in which interval arithmetic can be used to alert us to instabilities in computer algorithms, roundoff error accumulation, and even the effects of hardware inadequacies. This paper is primarily tutorial.

AMS(MOS) Subject classification: 94-04

Keywords: interval arithmetic
roundoff error
error control
error detection

Work unit: #8 (Computer Science)



Sponsored by the United States Army under Contract No. DAAG29-75-C-0024 and a grant from the Waterways Experiment Station.

APPLICATION OF INTERVAL ANALYSIS TO ERROR CONTROL

J. M. Yohe

1. Introduction.

Interval analysis, far from being just an academician's toy, provides a viable means for uncovering sources of potential error in digital computations. In this note, we examine some of the ways in which interval analysis can be employed to enhance the accuracy of computations, whether or not it is employed in the production environment; we also suggest guidelines which can help determine whether interval analysis should be used in the production situation.

The basic reference for interval analysis is the book by R. E. Moore [3]. For the sake of completeness, however, we will sketch the theory very briefly.

A (real) interval $[a, b]$ is defined in the usual fashion:

$$[a, b] = \{ x \mid a \leq x \leq b \} .$$

A real-valued function f of a real variable can be extended to an interval-valued function of an interval variable by defining

$$\bar{f}([a, b]) = \{f(x) \mid x \in [a, b]\} ,$$

assuming, of course, that f is defined and continuous on $[a, b]$. Real-valued functions of several real variables are similarly extendable, and we can thus define the usual arithmetic operations on intervals (except, of course, for division by an interval containing zero).

Intervals can be regarded as representing real numbers where some uncertainty is present. Degenerate intervals (intervals whose left endpoint

Sponsored by the United States Army under Contract No. DAAG29-75-C-0024 and a grant from the Waterways Experiment Station.

is the same as the right) would then represent exact real numbers; indeed, interval operations and functions on degenerate intervals are the same as their real counterparts. We thus identify the real numbers with degenerate intervals, and we note that the interval number system is a *bona fide* extension of the reals.

In computing, of course, the ideal is seldom attainable. Even when we begin by using exact data, uncertainty creeps in due to roundoff error. Since the result of applying a given function to machine representable arguments is often not machine representable, it rapidly becomes necessary to work with approximations. Depending on several factors, including hardware behavior, algorithm stability, and data, the approximation will be more or less accurate.

In interval analysis, roundoff error is handled by making certain that the left endpoint of the interval is always rounded down, and the right endpoint is always rounded up. A formal treatment of interval analysis on finite-precision number spaces may be found in Kulisch [1].

In the case of ordinary computer arithmetic, roundoff error may accumulate to the point where the result of the computation has no significance. In interval arithmetic, however, accumulated roundoff error will cause the intervals to become wide, but the interval will always contain the true result. Thus the user can get some idea of how stable the algorithm is with the given data, and how uncertain the answers may be.

In the following sections, we give some examples of ways in which interval analysis can reveal instabilities in algorithms or weaknesses in

hardware.

The examples we give were run on the UNIVAC 1110 at the University of Wisconsin using ordinary single precision floating point arithmetic and interval arithmetic as implemented in the INTERVAL II package of the Mathematics Research Center. Results are given in octal, since the 1110 is a binary machine; however, since the octal number system is foreign to most of us, approximate decimal values are given for reference.

2. Summation

In elementary numerical analysis, one learns that the order in which terms of a summation are added can affect the accuracy of the result. To illustrate this, we considered the alternating sum

$$y = \sum_{i=0}^{128} (-1/5)^i .$$

We summed this series in both directions; i.e., with the largest term first (from the left) and with the smallest term first (from the right). The results are shown in Table 1. Note that the sum of the corresponding infinite series is 5/6, and that the correctly rounded 9-digit octal fraction corresponding to 5/6 is .652525253. While the 1110 floating point arithmetic produced results which look reasonably good, note that the better of the two differs from the correctly rounded value by one bit in the low-order position. (The fact that it does not even lie in the interval computed by the Interval II package is due to a hardware anomaly: see Section 4 for details.)

The interval arithmetic results are more revealing. When we sum the series from the right, we obtain bounds that are best possible; i.e., the upper and lower bounds differ by just one bit in the least significant position.

TABLE 1

Summation of $\sum_{i=0}^{128} (-1/5)^i$

Summation beginning with i =		Interval	1110 floating point
0	octal	(.652 525 154, .652 525 354)	.652 525 255
	decimal	(.83333 28644, .83333 38201)	.83333 33507
128	octal	(.652 525 252, .652 525 253)	.652 525 254
	decimal	(.83333 33284, .83333 33358)	.83333 33433

Summing from the left, however, the upper and lower bounds differ by a whopping $200|_8$. In the light of the fact that the 1110 produced a much closer answer using standard floating point arithmetic, this might seem to call interval arithmetic into question. However, it would not have been difficult to produce a collection of terms whose true sum would have been very close to one extreme or the other, and yet would have given the same result as the above numbers gave on the 1110. Since the numbers involved in a real-life computation are seldom as clean as the ones in our example, the width of the interval turns out to be a strength of interval arithmetic rather than a weakness; it clearly reveals the advantage to be gained by careful attention to the order of summation. In many cases, the relative sizes of the terms of a summation will not be known in advance (as they were here); in such cases, interval arithmetic can give an excellent idea of the possible roundoff error involved in the summation.

3. Roots of a quadratic equation:

One of the simplest examples of how seriously roundoff error can affect the results of a computation is the classic example of solving for the roots of the equation $ax^2+bx+c = 0$. In the case where 4ac is small compared to b^2, so that the discriminant is very nearly b^2, the relative error in the root with smaller absolute value can be very high indeed if the root is calculated by blind application of the quadratic formula. This is not a new fact; indeed, it is discussed in many elementary numerical analysis texts. The fix is simple; one calculates the root with larger absolute value (the one using addition in the numerator of the quadratic formula if b is negative, or subtraction if b is positive) using the quadratic formula, then finds the other root by dividing the known root into c/a.

Naturally, the errors involved are the most dramatic when the coefficients of the quadratic equation are close to the limits of the hardware's ability to produce results with a modicum of accuracy. For purposes of illustration, we took

$$a = 2^{-29}$$

$$b = -1$$

$$c = 1.$$

We solved for both roots using the quadratic formula, and then solved for the root with smaller absolute value using the above algorithm. The results are shown in Table 2.

Several observations are in order here. The floating point arithmetic on the 1110 gives a result of zero for the second root using the quadratic

TABLE 2

Roots of a quadratic equation

Root		Interval	1110 floating point
Larger	oct	(3 777 777 774, 4 000 000 000)	4 000 000 000
	dec	(5368 70908, 5368 70912)	5368 70912
Smaller, by quadratic formula			
	oct	(0.000 000 000, 2.000 000 000)	0.000 000 000
	dec	(0, 2)	0
Smaller, by division			
	oct	(1.000 000 000, 1.000 000 002)	1.000 000 000
	dec	(1.00000 00000, 1.00000 0015)	1.00000 00000

formula. It is quite easy to check that this is nonsense; substituting this back into the original equation gives the equation $1 = 0$. But in many computations, the result of one calculation is simply used in a further calculation without being checked; the error is not discovered until later, if at all. The interval analysis result is the interval (0,2), which is almost uselessly wide -- but the width of the interval points out the instability of the algorithm for the given data.

Passing to the more accurate results, note that here also the interval result gives more information. The standard arithmetic states flatly that the larger root is 2^{29}; but again, substituting this in the original equation gives the result $1 = 0$. To an unsophisticated program, this root might appear to be just as questionable as the value of zero obtained by the quadratic formula for the other root. But the interval analysis package tells us that the true

root lies somewhere between $2^{29}-4$ and 2^{29}; these are the closest bounds available on the 1110. That is, the calculation has been performed with the 1110's maximum accuracy. Likewise, the smaller root is shown to be calculated with maximum accuracy; and indeed, the interval package lets us deduce that the root is greater than or equal to one, if we are interested in knowing that. The ordinary floating point arithmetic leaves us to guess which side of 1 the root lies on.

4. Hardware Limitations:

Unfortunately, present-day hardware is not always as well-designed as it could be; proper rounding is often not available. This can result in unwarranted error accumulation. The following example, which we ran on the UNIVAC 1110, illustrates this rather dramatically.

We computed $X(1) := 1.0 - 2^{-27} - 2^{-27}$. On the 1110, this is not the same as $1.0 - 2^{-26}$, since the 1110 carries a 27-bit fraction in its floating point representation, and truncates after equalizing exponents but before executing the add or subtract operation. The result of this is that, although $1.0 - 2^{-27}$ is representable on the 1110, it can not be computed by performing the indicated subtraction.

Having computed $X(1)$, we computed $X(I) := X(I-1)^2$ for $I := 2, 3, \ldots,$ 40. Finally, we summed the $X(I)$'s from both left and right. The results, both in interval arithmetic and in standard 1110 arithmetic are shown in Table 2. We have shown the octal results in standard 1110 floating point format; the first three octal digits are the exponent part of the number and the last nine are the fractional part. The exponent is the appropriate power of 2,

TABLE 3

Term	Decimal Interval Left	Decimal Interval Right	Decimal 1110	Octal Interval Left	Octal Interval Right	Octal 1110
1	.9999999851+00	.9999999851+00	.1000000000+01	200777777776	200777777776	201400000000
2	.9999999702+00	.9999999776+00	.1000000000+01	200777777774	200777777775	201400000000
3	.9999999404+00	.9999999627+00	.1000000000+01	200777777770	200777777773	201400000000
4	.9999999808+00	.9999999329+00	.1000000000+01	200777777760	200777777767	201400000000
5	.9999997616+00	.9999998733+00	.1000000000+01	200777777740	200777777757	201400000000
6	.9999995232+00	.9999997541+00	.1000000000+01	200777777700	200777777737	201400000000
7	.9999990463+00	.9999995157+00	.1000000000+01	200777777600	200777777677	201400000000
8	.9999980927+00	.9999990389+00	.1000000000+01	200777777400	200777777577	201400000000
9	.9999961853+00	.9999980852+00	.1000000000+01	200777777000	200777777377	201400000000
10	.9999923706+00	.9999961779+00	.1000000000+01	200777776000	200777776777	201400000000
11	.9999847412+00	.9999923632+00	.1000000000+01	200777774000	200777775777	201400000000
12	.9999694824+00	.9999847335+00	.1000000000+01	200777770000	200777773777	201400000000
13	.9999389648+00	.9999694750+00	.1000000000+01	200777760000	200777767777	201400000000
14	.9998779297+00	.9999389574+00	.1000000000+01	200777740000	200777757777	201400000000
15	.9997558743+00	.9998779222+00	.1000000000+01	200777700002	200777737777	201400000000
16	.9995116007+00	.9997558668+00	.1000000000+01	200777600013	200777700001	201400000000
17	.9990231824+00	.9995117933+00	.1000000000+01	200777400065	200777600012	201400000000
18	.9980466110+00	.9990238249+00	.1000000000+01	200777000351	200777400064	201400000000
19	.9961010292+00	.9980486035+00	.1000000000+01	200776001721	200777000350	201400000000
20	.9922172576+00	.9961010218+00	.1000000000+01	200774007632	200776001720	201400000000
21	.9844950810+00	.9922172512+00	.1000000000+01	200770037365	200774007631	201400000000
22	.9692305624+00	.9844950736+00	.1000000000+01	200760175764	200770037364	201400000000
23	.9394078776+00	.9692305550+00	.1000000000+01	200740764107	200760175763	201400000000
24	.8824871555+00	.9394078702+00	.1000000000+01	200703652555	200740764106	201400000000
25	.7787835747+00	.8824871480+00	.1000000000+01	200616571342	200703652554	201400000000
26	.6065034517+00	.7787835672+00	.1000000000+01	200466417261	200616571341	201400000000
27	.3678469211+00	.6065034517+00	.1000000000+01	177570531564	200466417261	201400000000
28	.1353113558+00	.3678469248+00	.1000000000+01	176425074173	177570531565	201400000000
29	.1830916243-01	.1353113614+00	.1000000000+01	173453764310	176425074176	201400000000
30	.3352254462-03	.1830916456-01	.1000000000+01	165537404625	173453764317	201400000000
31	.1123760995-06	.3352255080-03	.1000000000+01	151742515520	165537404646	201400000000
32	.1262838774-13	.1123761413-06	.1000000000+01	122706770506	151742515577	201400000000
33	.1594761762-27	.1262839716-13	.1000000000+01	044624243723	122706770637	201400000000
34	.0000000000	.1594764154-27	.1000000000+01	000000000000	044624244162	201400000000
35	.0000000000	.1469367939-38	.1000000000+01	000000000000	000400000000	201400000000
36	.0000000000	.1469367939-38	.1000000000+01	000000000000	000400000000	201400000000
37	.0000000000	.1469367939-38	.1000000000+01	000000000000	000400000000	201400000000
38	.0000000000	.1469367939-38	.1000000000+01	000000000000	000400000000	201400000000
39	.0000000000	.1469367939-38	.1000000000+01	000000000000	000400000000	201400000000
40	.0000000000	.1469367939-38	.1000000000+01	000000000000	000400000000	201400000000
Left sum	.2566712332+02	.2666712904+02	.4000000000+02	205632531046	205652531076	206500000000
Right sum	.2566712403+02	.2666712570+02	.4000000000+02	205632531051	205652531060	206500000000

biased by 200_8 .

The table is self-explanatory. However, we can not help noting that, even in the worst case, the interval arithmetic package gives answers that are far better than the 1110's floating point arithmetic yields; the true answer has to lie somewhere between 25.6 and 26.7; yet the 1110 computes 40!

The 1110 is not unique in its foibles; many other computers have anomalies in their arithmetic operations. We even know of one series of computers in which 1.0 is not a multiplicative identity, owing to truncation prior to post-normalization!

5. Analysis of algorithm limitations:

A good background in numerical analysis will enable one to avoid many of the obvious sources of numerical error, such as the ones we have displayed in Sections 2 and 3. However, even the best of algorithms may be somewhat data sensitive, and other algorithms in common use may not be the best that can be devised.

The reliability of a given algorithm applied to a reasonable data set may be approximated through the use of interval analysis. The interval version of the algorithm should be run using several representative data sets, and the widths of the resulting intervals examined. If the algorithm produces acceptably narrow intervals for several representative data sets, it is likely that the algorithm is stable. Interval results which are unacceptably wide do not necessarily indicate a bad algorithm, but simply alert one to the possibility that data sensitivity may exist.

One of the most common sources of pessimistic interval bounds is the repeated use of the same variable in a mathematical formula. As an extreme example, consider $y = x/x$. Clearly the result should be exactly 1, and would be in most if not all floating point arithmetic systems. But if x is the interval $(\frac{1}{2}, 3)$, then y would be the interval $(1/6,6)$ -- a far cry from $(1, 1)$! The trouble is, of course, that the interval arithmetic routines do not recognize the dependence.

Nevertheless, if a computation produces intervals of unacceptable width, it is a signal that the computation should be scrutinized for unstable operations. It may be necessary to insert several checkpoints to ascertain where the computation loses accuracy. Once the trouble spot is pinpointed, there are several things that one may consider in attempting to solve the problem:

1. Is there an equivalent, but more stable, algorithm which can be used? The square root example illustrates this possibility.

2. Is there a critical summation which could be made more accurate by judicious choice of the order of summation? It might even be worth invoking a rather sophisticated sort-and-sum algorithm to perform the summation.

3. Can the critical portion be rewritten using higher precision arithmetic in such a way as to improve accuracy?

The latter question may well point the way to research problems. Not enough attention has been paid to the problem of what is required within an algorithm in order to produce a result of a given precision. Kulisch [2] has stated that inner products can be formed to full single precision accuracy

using an accumulator that is only slightly longer than double precision; however, for many types of problems results of this kind are unknown.

6. Analysis of system limitations:

The example of Section 4 points out what should have been obvious from the beginning: if interval analysis is to be used for the purpose of analyzing a system (hardware and/or software), then its performance must be at least as bad as the performance of the system under study. INTERVAL II was designed as a production package, and wherever feasible its operations yield results of the greatest possible accuracy. Thus, in its present form, it can not be used effectively to predict the effects of hardware and software anomalies on the results of a computation.

If this sort of analysis is required, the INTERVAL II package can be modified to handle it. One would simply use as a basis the data type under study, supply error bound information on the mathematical function routines, and rewrite the five arithmetic primitives to emulate the action of the hardware and at the same time obtain rigorous bounds on the results thus obtained. In particular, the upper bound for an operation would need to be the greater of the upper bound obtained by rounding the true result up and the result actually obtained by the system under study; the lower bound would be treated analogously.

Assuming that the INTERVAL II package has been thus modified, appropriate test programs should be run using the modified INTERVAL II, and the results compared with the results obtained when the unmodified INTERVAL II package was used. The difference is the amount by which system inaccuracies could affect the accuracy of the results.

7. Interval analysis as a production tool:

In cases where accuracy is of critical importance and instability of the algorithm in certain situations can not be eliminated, it may be necessary to verify some or all of the computation by using interval arithmetic. At present, there is indeed a penalty associated with doing this; the arithmetic operations are likely to take an order of magnitude or two more time in interval arithmetic than in standard floating point. This would not need to be the case if hardware were designed to be more hospitable to interval arithmetic; details are given in [4]. Surprisingly, the standard mathematical functions in interval arithmetic are not that much more expensive; here the factor is two to three times as long. Even though it is more costly in terms of machine time, it may be worth it: technology is driving hardware prices down to the level where inefficiency is no longer regarded as an unpardonable sin, and the extra cost may seem small when balanced against possible failure of a structure designed using the results of computation.

8. Conclusions:

We have given simple examples of the ways in which computer algorithms can be data sensitive and the effects that roundoff and truncation error can have on computations. We have also pointed out how a seemingly insignificant design lapse in one production computer can lead to ridiculous results in an admittedly contrived, but still simple, problem. Finally, we have given a few elementary guidelines for the application of interval arithmetic to the detection and control of error in both hardware and software.

Certainly, the error encountered in a computation will seldom be as great as might be indicated by the results of performing the computation in interval arithmetic -- but it might be. It could be said that a given bridge will seldom collapse, for having once done so it no longer exists and therefore cannot collapse again. But once is once too often, both in terms of bridges collapsing and of computers producing trusted answers that are not even in the right ball park. And if such simple problems as we have presented can lead to such dramatically wrong answers, the mind boggles at what could happen in the sorts of complicated calculations that we routinely use in making critical decisions today.

References

1. U. Kulisch, An axiomatic approach to rounded computations, Numer. Math. 18 (1971), 1-17.

2. U. Kulisch and G. Bohlender, Formalization and implementation of floating-point matrix operations, Universität Karlsruhe report, Sept., 1974 (To appear in Computing).

3. Ramon E. Moore, Interval Analysis, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1966.

4. J. M. Yohe, Roundings in floating-point arithmetic, IEE Trans. Computers C-22 (1973), 577-586.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: 1686	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER: (2) Technical	
4. TITLE (and Subtitle): APPLICATION OF INTERVAL ANALYSIS TO ERROR CONTROL,	5. TYPE OF REPORT & PERIOD COVERED: Summary Report, no specific reporting period
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): (10) J. M. Yohe	8. CONTRACT OR GRANT NUMBER(s): (15) DAAG29-75-C-0024
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Mathematics Research Center, University of Wisconsin, 610 Walnut Street, Madison, Wisconsin 53706	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: See Item 18	12. REPORT DATE: September 1976
	13. NUMBER OF PAGES: 13
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): (12) 16p.	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(14) MRC-TSR-1686

18. SUPPLEMENTARY NOTES

U.S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

and

Waterways Experiment Station
Vicksburg, Mississippi 39180

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

interval arithmetic
roundoff error
error control
error detection

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We give simple examples of ways in which interval arithmetic can be used to alert ~~us to~~ instabilities in computer algorithms, roundoff error accumulation, and even the effects of hardware inadequacies. This paper is primarily tutorial.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

221200

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)